Microsoft® Collaboration with Exchange 2000

PT107
Draft 0.5

From the Exchange 2000 Server Pre-emPt Series

Peter Lamsdale
Microsoft Consulting Services
United Kingdom

Microsoft Confidential

© 1999 Microsoft Corporation. All rights reserved.

The information contained in this White Paper represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

BackOffice logo, Microsoft, Win32, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries..

Other product or company names mentioned herein may be the trademarks of their respective owners.

Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA

Contents

Acknowledgments

Many thanks to the following people for their time and enthusiasm. No whitepaper is produced without many people going out of their way to assist.

Exchange Product Group

Doug Wood

Andrew Sinclair

Jim Reitz

Jason Mayans

Microsoft Consulting Services UK

Paul Bowden

Darren Strange

Document Vision / Scope

The purpose of this document is to provide the reader with a better understanding of new collaboration features that “Platinum” provides for developers.

After reading this document you should understand the fundamentals of Collaborative Data Objects (CDO), Collaborative System Management Objects and the usage of ADO/OLEDB against the “Platinum” store, and the relationship between these programming models. You will also understand the direction in which Microsoft is evolving these technologies.

This document will introduce and position these technologies providing some code samples to help your understanding. For more detailed information review the CDO SDK and supporting white papers.

This information should be part of the learning cycle for Exchange “Platinum”. To gain a full understanding of the other technologies in the product, you may wish to read the other documents within this series:

- PT101 - Deploying the Active Directory Connector
- PT102 - “Platinum” Directory Access & Integration with Windows 2000
- PT103 - Exchange 5.5 and “Platinum” Co-existence & Migration
- PT104 - Understanding “Platinum” Message Routing
- PT105 - Understanding the “Platinum” Store
- PT106 - Deploying Exchange Data Conferencing Services

PLEASE NOTE: The information contained within this document is based upon Windows 2000 Beta 3 (build xxxx) and Exchange “Platinum” build 2007. This document will be updated for subsequent releases of these products.

Acronyms used in this document

CDO	Collaborative Data Objects
EDK	Exchange Developers Kit
COM	Component Object Model
CDONTS	CDO for Windows NT Server 4.0
MAPI	Messaging Application Program Interface
ADSI	Active Directory Services Interface
ADO	ActiveX Data Objects
MIME	Multi-purpose Internet Mail Extensions
WSH	Windows Scripting Host
MTS	Microsoft Transaction Server
SEO	Server Extension Objects
EOM	Exchange Object Model

Introduction

Collaboration on Exchange 5.5

There are many successful Microsoft Exchange Server 5.5 collaborative applications in use today. These applications have been written using a wide range of technologies and development tools. Developers have used OLE Messaging, Active Messaging, CDO, CDONTS, Outlook, Electronic Forms Designer(EFD), Active Directory Services Interface (ADSI), Simple MAPI and Extended MAPI to build their applications. These technologies have been manipulated with tools such as Microsoft Visual C++, Microsoft Visual Basic, Active Server pages and Microsoft Office. The features of these technologies have enabled a variety of applications to be written but to succeed developers have had to work with multiple object models and interfaces, and often with multiple development tools. Developers who have worked on traditional database systems have found the Exchange development interfaces immature in comparison to those for working with SQL Server, for example filtering and report writing have historically been more difficult to achieve with Exchange Server.

Exchange development has traditionally been centered around MAPI (messaging Application Programming Interface) which was the programming interface through to Exchange information. MAPI has become increasingly proprietary to Microsoft Exchange and Outlook and doesn't provide a platform for future extensibility to support all of the necessary internet protocols and standards, it also uses a unique database interface which doesn't fit in with the Microsoft universal data access methodology around OLEDB and ActiveX Database Objects (ADO). MAPI was designed as messaging middleware and has no inbuilt support for contacts, scheduling or document management.

Although the Exchange Developers Kit (EDK), which shipped in the Platform SDK, has provided developers with some useful utilities, such as creating mailboxes and controlling Folder security, where the various object models have not exposed the required functionality, there is a requirement for a more future proof approach. This is delivered in "Platinum".

A Brief history of CDO

CDO 1.21, formerly known as both OLE Messaging and Active Messaging, is a COM interface built on top of MAPI. This Interface enabled developers to access information on Microsoft Exchange Server and build collaborative applications whilst being able to use the same code on client and server machines. There were two CDO libraries in version 1.21:

- *CDO Library*

  This enabled developers to send and receive messages and interact with Exchange functionality such as Folders and Address books. It provided hierarchical access to the Exchange store and was capable of working with or without profiles on the local machine.

- *CDO Rendering Library*

  This library was designed to render Exchange information into HTML for publication on the Intranet/Internet. Outlook Web Access was a very good example of an application that used the rendering library.

There was also another form of CDO called Collaboration Data Objects for NT Server (CDONTS) which was designed to be used on Windows NT Server 4.0 with the SMTP

capabilities of Microsoft Internet Information Server (IIS) version 4.0. CDONTS was not reliant on MAPI and could work without Exchange Server. CDONTS was a subset of CDO 1.2 and was designed for use by developers who wanted the ability to incorporate fast large-scale messaging into their applications without the need for advanced functionality such as calendaring, scheduling or workflow. This server-only COM interface was excellent for use in Active Server pages making sending messages as simple as the code overleaf.

```
Set objMail = Server.CreateObject("CDONTS.Newmail")
objMail.Send ("rob@here.com", "jdr@there.com", "Bad news", "I owe you
£50")
Set objMail = Nothing
```

Code Sample 1 - Sending a message with CDO NTS

Knowledgebase article Q177850 provides a comprehensive functional comparison of CDO 1.2 and CDO for NTS.

CDO 1.21 and the Outlook Object Model

Microsoft Outlook 98 has a very comprehensive object model which enables developers to build collaborative applications either through automation or by using Outlook forms. It is possible to automate Outlook from other development tools so applications can use features of the Outlook product within their custom applications. Outlook is a client application and automating it is an option on the users machine but although the code can be used on the server an application could not scale and there were problems with security and profiles. So why would developers use the Outlook Object model at all if it is client–side only and required more memory?

The answer was that CDO 1.21 and the Outlook Object model had many common features such as being able to manipulate folders, send messages etc. but there were also unique features in the Outlook object model that didn't exist in CDO. A good example of this was that Outlook enabled developers to open up someone else's calendar and add appointments to their diary, the diary and the free/busy information would be automatically updated. With CDO this could not be done, developers would have to send an appointment to a mailbox and only when the user read or accepted the appointment would the free/busy information be updated.

The difference in functionality of the object models meant developers had to know which object model supported which features. Writing code for the Outlook object model used a totally different syntax to CDO. Although there were some attempts to help developers by adding functions to integrate the two object models, developers still had to use two object models in some applications.

Collaboration on “Platinum”

CDO in “Platinum” will have a object model that has been extended to represent the entire spectrum of collaborative data with interfaces for appointments, contacts and workflow aswell as messaging. The CDO interfaces have been extended to integrate with OLEDB / ADO and ADSI which will enable developers to integrate these technologies seamlessly into their collaborative applications. The many previous flavors of CDO interfaces are to be unified by CDO in “Platinum”. CDO in “Platinum” is not based on MAPI. CDO alone will not replace MAPI but with the integration provided for OLEDB / ADO and ADSI it will offer a complete solution to developers.

Microsoft’s universal data access methodology is centered around OLEDB / ADO. These interfaces offer excellent access to data in tabular formats such as Microsoft SQL Server and other ODBC/ISAM databases but Exchange Server information is held in a variety of structures including documents. The OLEDB for Docs interface enables developers to gain access to document schemas such as MIME based contents like iCalendar and VCard. The new CDO object model conforms to Microsoft’s universal strategy by taking the best of messaging and the best of methodology for data access and coupling the two interfaces very tightly.

There will be two flavors of CDO which are explained *overleaf.*

CDO for Windows 2000

Collaboration Data objects for Windows 2000 is a COM component that enables developers to build applications that create, send and manipulate messages in plain text or internet formats. The CDOSYS.DLL library is an integral part of the Windows 2000 series of Operating systems. It is installed by default with Windows 2000 workstation and Windows 2000 Server. Both the workstation and Server include an SMTP Service so developers have built in support to build messaging applications. Windows 2000 Server also provides an NNTP Service. The SMTP and NNTP services do not provide support for mailboxes.

The Library supports dual interfaces enabling the widest range of development tools to be used. CDO is designed to work on both the client and the server. The library supports internet standards such as Multi-Purpose Mail Extensions (MIME) which enable messages to be structured with more complex data[1]. It is possible to send messages using both the SMTP and NNTP protocols. Below is a list of example applications that could be written with this library.

- Creating and sending bulk email
- Manipulating messages as they pass through the server with protocol events[2]. This is useful for adding disclaimers to message contents, discarding junk mail, and filtering or redirecting messages.
- Checking inbound messages for Viruses

CDO for Exchange

Collaboration Data Objects for Exchange is a COM component designed to provide developers with the ability to create rich collaborative applications that take advantage of the Microsoft Windows 2000 and Microsoft Exchange platforms. The CDOEX.DLL library can be used to create applications that provide such features as messaging, calendaring, contacts, store events and Public Folders. The CDO object model has been designed to integrate with the ActiveX Data Objects library version 2.5. This integration provides a

[1] See Understanding Message Formats later in this document.
[2] See Exchange Server protocol events later in this document

consistent data access interface to the Exchange Store and the Windows 2000 directory and is a major enhancement from the previous version of the library. The OLE DB / ADO integration enables developers to access Exchange store information using SQL queries and manipulate it with ADO Recordset and Record objects.

This library also supports dual interfaces enabling Microsoft Visual C++ developers the ability to use low-level OLE DB interfaces to access the Exchange store through the OLE DB 2.5 provider whilst not limiting access to Visual Basic, Visual J++ and Script developers.

The Exchange store is a database of messages, newsgroup posts, appointments, contacts etc. which can all be manipulated through the CDO Object model. The object model has been extended to manipulate all these objects aswell as Folders, Calendars and Exchange Management objects. Messages in the Exchange store are held in MIME format and as the type of data held in a folder is not always obvious the MIME headers of messages can be examined to determine the content-class.

It is possible to create richer applications with this library than with the CDO for Windows 2000 Library. This library can create the applications list above and can also create the following types of applications

- Applications that manipulate calendars and create appointments/meetings. Recurring Calendar events with exception handling.
- Contact Management Systems
- System administration functions such as mailbox creation and maintenance, folder manipulation and quota limits.

CDO for Windows 2000 and CDO for Exchange are not code-compatible with the CDO 1.2.1 libraries. Any code solutions written using previous versions of CDO will still work on machines where the library is installed but the code will need to be re-written for it to use the new CDO object model. The different versions of the libraries will co-exist on the same machine. It is possible for an application to call difference versions of the CDO library so old applications can be extended by adding the new CDO libraries into the project.

The future of MAPI

MAPI has enabled many developers to create successful messaging applications but the advent of internet industry standards has meant that MAPI has become increasingly more proprietary to Microsoft. Industry standards such as LDAP, POP, IMAP and NNTP have provided a improved platform for collaborative development. MAPI and the CDO 1.2 COM interface that use it are being retired. Microsoft will do no further development work with MAPI and will remove all traces from it in the future from the Exchange and Outlook product lines. MAPI clients will be supported for the foreseeable future to ensure backward compatibility.

Below are some of the reasons why MAPI will be phased out.

- MAPI provides no support for MIME messages
- There is no support for collaborative features such as calendaring, contacts or document management
- MAPI’s message format is Rich Text Format (RTF) and not the industry standard, HTML
- MAPI uses a proprietary database that does not use OLE DB
- MAPI is difficult to port to other platforms

1.

Messaging with CDO

The CDO object library in “Platinum” provides the developer with many abilities including the simple task of sending email messages from their applications. It is possible to send simple messages with just a few lines of code but there are some new features of the CDO object model that have enabled more complex operations to be carried out. This section looks at some code examples that demonstrate some of the messaging behaviors available to the developer.

Sending a simple mail message

Sending a message is a fundamental part of most messaging applications and CDO makes this task very simple. The sample below creates a new message with an attached Microsoft Word document. Attachments are added individually by calling *AddAttachment*. This method can be called as many times as is required and it supports the use of fully qualified directory names and UNC paths, as well as "http://", "https://", "ftp://", and "file://" prefixes. The message is submitted using the default delivery options. Configuration information can be set specifically for a message or CDO will provide default settings.

```
Dim oMessage As CDO.Message
Set oMessage = New CDO.Message
With myMessage
    .From = "Darren Strange <DStrange@Disney.com>"
    .To = "Paul Bowden <PBowden@Microsoft.com>"
    .Subject = "This was sent to you from CDO 2.0"
    .TextBody = "Great isn’t it!"
    .AddAttachment "file://c:/Autoexec,bat"
    .Send
End With
```

Code Sample 2 - Sending a message with CDO

Configuring how the mail is sent

It is possible to use the default mail delivery configuration for messages or developers can configure message delivery through code. CDO has provided a new object called a configuration object that enables this behavior. There are a lot of fields that can be configured for SMTP and NNTP transport and these fall into the http://schemas.microsoft.com/cdo/configuration namespace. The example below shows how to set some simple SMTP settings to set the SMTP and proxy server and the ports that are used. The configuration object is later attached to the message.

```
Private Sub SetConfigurationSettings()
    Dim oConfiguration  As CDO.Configuration
   'Set the Message Configuration
    Set oConfiguration = CreateObject("CDO.Configuration")
   'The following lines should all be on line each
oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
urlproxyserver") = "itgproxy"
```

```
oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
proxyserverport") = "80"
Configuration.Fields("http://schemas.microsoft.com/cdo/configuration/
smtpsserver") = "INET-IMC-02"
oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
smtpserverport") = "25"
    oConfiguration.Fields.Update
End Sub
```

Code Sample 3 - How to use the CDO Configuration object

Sending HTML pages in a message object

There is a new method of CDO called 'CreateMHTMLBody' that enables developers to create mail messages that contain the contents of a web page. Below is a code example showing how this can be achieved.

The sample shows how the configuration object can be used to manually configure the settings of the proxy server, SMTP server and the respective ports to be used in sending the message. The contents of the front page from the English Financial Times newspaper is sent as a message to the user specified in the message properties.

```
Private Sub cmdSendWebPageAsEmail_Click()
   'Procedure level variables
    Dim oConfiguration  As CDO.Configuration
    Dim oMessage        As CDO.Message
   'Set the Message Configuration
    Set oConfiguration = CreateObject("CDO.Configuration")

oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
urlproxyserver") = "itgproxy"

oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
proxyserverport") = "80"

oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
smtpserver") = "INET-IMC-02"

oConfiguration.Fields("http://schemas.microsoft.com/cdo/configuration/
smtpserverport") = "25"
    oConfiguration.Fields.Update
   'Instantiate the outgoing message object
    Set oMessage = CreateObject("CDO.Message")
    Set Message.Configuration = oConfiguration
    With oMessage
        .From = "WebPage-Mailer <Web@Web.com>"
        .To = "Peter Lamsdale<peterlam@microsoft.com>"
        .Subject = "WebPage: Financial Times"
        .CreateMHTMLBody "http://www.ft.com/hippocampus", 0
        .Send
    End With
   'Close down all Object References
    Set Message = Nothing
    Set Config = Nothing
```

1.

```
End Sub
```

Customizing Message Headers

Some message headers are required by the SMTP server for message delivery. Others have special meaning to the messaging application that sent the message.

Additional message headers can be created by adding resource fields to the Fields collection. Resource fields typically include properties that are available through Bodypart or Message object properties, but can include other properties that are not handled directly by CDO.

You can add custom headers using the fields collection of a message. The MIME specification states that any custom headers should be prefixed with 'x-'. In CDO each x-header must be part of the 'urn:schemas:mailheader' namespace. This means you must prefix your x-header with the 'urn:schemas:mailheader:' string as shown in the example below.

The example below shows a message being sent with a custom header called 'x-Custom'. The application that receives this message will need to know how to process the custom header.

```
Private Sub SendCustomMessageHeaders()
    Dim oMessage As CDO.Message
    Set oMessage = CreateObject("CDO.Message")
    With oMessage
       .Fields("http://www.iana.org/standards/rfc822/Importance") = "high"
       .Fields("http://www.iana.org/standards/rfc822/x-Custom") = "My
Data"
       .From = "peterlam<peterlam@Microsoft.com>"
       .To = "Paul Bowden<PBowden@Microsoft.com>"
       .Subject = "Memory Loss"
       .TextBody = "..."
       .Send
   End With
End Sub
```

Code Sample 4 - Sending custom message headers with CDO

Posting a news Item

Messages that are sent to newsgroups are the same CDO Class ,CDO.Message, as normal SMTP messages except that they have different headers that relate to newsgroups and the delivery process is not to a user mailbox. The newsgroup-related header properties are Newsgroups, Keywords, FollowUpTo, and Organization. In order to post a message, you must specify at least one newsgroup name in the Newsgroups property. Messages that are to be sent to multiple newsgroup names require that the names are separated by commas. News items are not addressed to individual recipients, but rather to an entire newsgroup so the 'Post' method is used as oppose to Send.

1.

The code below shows how simple it is to create a new post in a newsgroup with CDO. The basis of the post is still a CDO message object. The Newsgroups property is set and the normal subject and textbody fields, this example also sends an image file. Finally the message is posted to the newsgroup.

```
Private Sub PostToNewsGroup()
    Dim myMessage As New CDO.Message
    With myMessage
        .From = "David Gower<DGower@Cricket.com>"
        .Newsgroups = "alt.binaries.pictures.Insects"
        .Subject = "The world’s biggest cricket"
        .TextBody = "We found it. Here is a picture"
        .AddAttachment "c:\Monster.gif"
        .MIMEFormatted = False
        .Post
    End With
End sub
```

Code Sample 5 - Sending a new post to a Newsgroup

Understanding Message Formats

CDO components have been redesigned to use internet standards for messaging, calendar exchange and contact management. In order to write applications using CDO it is useful to understand the way in which messages are internally constructed. In the simplest scenario, a mail message has two parts - a body and a header. The purpose of the header is tell the email system how to route the message, how the message is encoded and how the receiving application should display the contents of the message whereas the body contains the actual content of a mail message.

Email messages that are sent over the internet are typically sent in a format defined in Request For Comment (RFC) 822 whereas messages sent to newsgroups over NNTP are sent in a defined in RFC 850. These different formats specify how the message is laid out internally. In order to send messages that are not just plain text requires the use of different formats such as MIME. MIME is used so that messages can be sent that contain formatting information, pictures, sounds or files. A MIME message can contain multiple content parts which can be arranged in a hierarchy. Whereas some MIME messages are simple others can be made up of many parts in a complex hierarchy. Figure 1 below is a diagram of how a MIME formatted message might be internally structured.

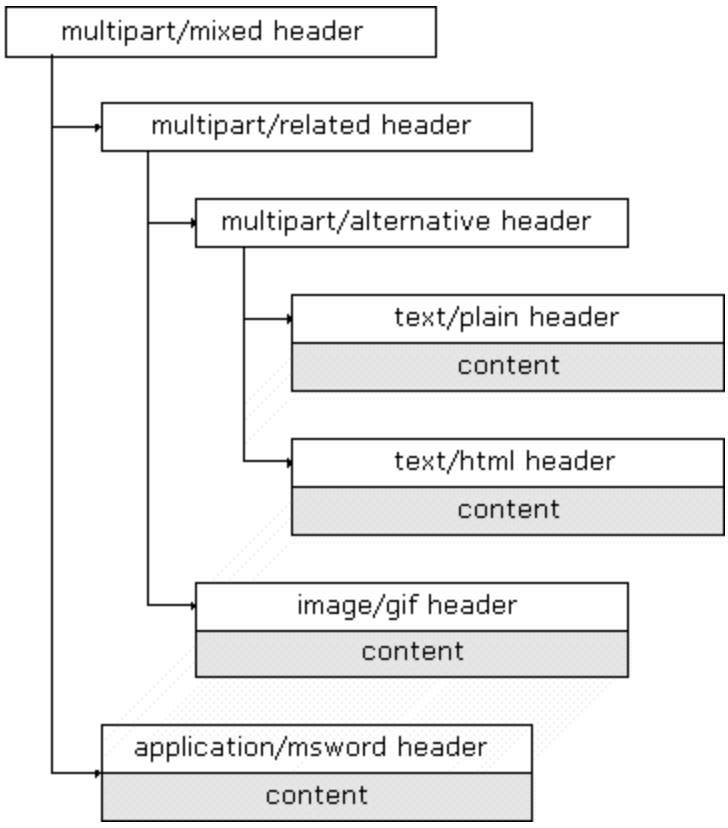

Figure 1 - An example of a the structure of a MIME formatted message

The diagram shows a message that contains many parts. The diagram shows the white areas as MIME headers and the shaded areas as message content. The structure is hierarchical and the headers are important in defining the structure.

1. The first header alerts the application that receives the message that the message contains different types of content
2. The third header indicates that the content sections are available in more than one format. There are two versions of the same content except that one is formatted as ‘text/plain’ and the other is formatted as ‘text/html’. The receiving application can then decide which format to display the message in based on user settings or the capability of the client.
3. The sixth header is of type ‘Image/gif’ which is an attachment that is an image file in *.GIF format. The imager file is used by the HTML content.
4. The Last header is of type ‘Application/MsWord’ which indicates that there is an attachment in the message that is a Microsoft Word document.

MIME can be passed over any mail transport system that is SMTP compliant. MIME is typically encoded over the wire, this enables MIME to handle many complex data types in its content Bodyparts. The receiving mail agent must simply decode the information for display. As MIME can contain both complex and simple ways of displaying the content information mail agents with limited capabilities can process the message whilst not compromising the capabilities of a very advanced system.

Understanding the format of MIME messages will help developers to relate the CDO object model to the message structure. The Message object in the CDO object model has a Bodyparts collection that can be manipulated. The code sample below shows how a message can be interrogated to find all the fields MIME headers in a message.

```
Public Sub GetRawMessageMime(oMessage As CDO.Message)
   'Procedure level variables
    Dim oField As ADODB.Field
    Dim oBodyPart As CDO.IBodyPart
    Dim oBodyPartField As ADODB.Field

   'Set up error Trapping for this routine
    On Error Resume Next

   'Loop through all the fields in the message object
    For Each oField In oMessage.Fields
        Debug.Print oField.Name & " " & oField.Value
    Next oField

   'Loop through the Bodyparts collection of a message
    For Each oBodyPart In oMessage.BodyPart.BodyParts
        For Each oBodyPartField In oBodyPart.Fields
            Debug.Print oBodyPartField.Name & " " & oBodyPartField.Value
        Next oBodyPartField
    Next oBodyPart
 End Sub
```

Code Sample 6 - Extracting the MIME Headers in a message

1.

The MIME specification is outlined in RFC2045 and can be seen on the web at the following URL http://sunsite.auc.dk/RFC/rfc/rfc2045.html.

1.

Calendaring with CDO

The CDO for Exchange Object model provides an interface which enables the ability to create and exchange calendar information between collaboration based systems. CDO is built on standard internet format such as iCalendar and iMIP, which enables appointments to be sent to a wider range of applications for processing.

The library provides support for appointments, meeting requests/responses, recurring appointments with exceptions, time zone support and Free/busy operations. Support for tasks will not occur in the “Platinum” release of Exchange Server.

The information in a meeting request or a meeting response is added into a CalendarMessage object which acts as an mechanism for sending an appointment item. CDO provides syntax for users to be able to receive and process meeting requests. A recipient can either accept or reject the appointment though code.

Typically an appointment or meeting request requires a response but there is a new CDO ‘Publish’ method that enables people to send out an appointment to many peoples calendars without the need for a reply. This is useful for a lot of scenarios where subscribers can receive information via notifications, a few example uses are listed below.

- posting corporate events such as training courses, company holidays or social events.
- A sports team submitting its fixture list to its supporters
- A list of milestones for a project

Create an appointment in the current user’s calendar

To update their own schedule a user must create appointments through the CDO Appointment object. This object exposes all the standard appointment fields as shown below in the example below. The appointment is saved into the address specified by the SaveTo method of the DataSource interface. The address format used for this is determined by whether the code is running on the machine where Exchange is located or from a remote machine on the network. The ‘FILE://:’ syntax is used when the code is running on the Exchange Server machine whereas from remote sites a HTTP URL is used. It is important to note that the address used with the SaveToContainer method must also include the name of the appointment being created. If the new name is the same as a name that has been previously used for a meeting at this time the new appointment will overwrite the previous one.

```
'**********************************************************************
' PROCEDURE    - CreateAppointment
' PARAMETERS   - None
' RETURN VALUE - None
' EXTERNALS    - Microsoft CDO library for Windows 2000
' DESCRIPTION  - Creates a new appointment
'**********************************************************************
  Private Sub CreateAppointment()
   'Procedure level variables
    Dim oAppointment   As New CDO.Appointment
    Dim oConfiguration As New Configuration
   'Set the Configuration properties of the Appointment object
    oConfiguration.Fields("TimeZoneID").Value = "Greenwich Mean Time :
Dublin, Edinburgh, Lisbon, London"
```

```
    With oAppointment
        Set .Configuration = oConfiguration
        .StartTime = #10/3/1999 10:00:00 AM#
        .EndTime = #10/3/1999 11:00:00 AM#
        .Location = "Atrium Building 2"
        .Subject = "Test Calendar Appointment"
        objAppt.DataSource.SaveToContainer _
        "file://./backofficestorage/ microsoft.com/MBX/user12/calendar"
    End With

 End Sub
```

Code Sample 7 - Creating an appointment in users own calendar

Send a meeting request to another user

Appointments are sent as MIME messages sent in the ICalendar format, the message is sent using a CalendarMessage with an appointment body part. Any application that can process MIME messages can process the appointment. The example below shows how to send a meeting request using the CDO Appointment object. A CDO configuration object is used to set the time zone, and the name and calendar location of the organiser of the meeting. When an appointment is sent the organiser of the meeting automatically has their own schedule updated with the meeting. The Free/busy time of the recipient of a meeting request is set to be tentative until the user reads the mail. This is very important as this was not the case in the previous version of CDO. There are many custom applications that need to update the calendars of other users such as sales representatives whose calendars are updated by their support staff, if a meeting request is sent to a user then there free/busy time reflects that they are already booked out, preventing double booking.

The properties of the appointment object are the same as the previous example except that in this example other attendees are asked to join the meeting through the CDO Attendees collection. Attendees can have a role specified which denotes whether they are the Chair, none required, required or optionally required to attempt the meeting.

```
'**********************************************************************
' PROCEDURE    - SendAppointment
' PARAMETERS   - None
' RETURN VALUE - None
' EXTERNALS    - Microsoft CDO library for Windows 2000
' DESCRIPTION  - Sends an appointment to another user.
'**********************************************************************
Private Sub SendAppointment()
   'Procedure level variables
    Dim oAppointment     As New CDO.Appointment
    Dim oCalendarMessage As New CDO.CalendarMessage
    Dim oAttendee        As New Attendee
    Dim oConfiguration   As New Configuration
   'Set the Configuration properties of the Appointment object
   'The cdoSendEmailAddress field is the meeting organizer
    With oConfiguration
        .Fields("TimeZoneID").Value = "Greenwich Mean Time : Dublin,
Edinburgh, Lisbon, London"
```

1.

```
        .Fields(cdoSendEmailAddress) =
"peterlam@finance.mcs.microsoft.com"
        .Fields("CalendarLocation") = _
        "file://./backofficestorage/ microsoft.com/MBX/Dstrange/calendar"
        .Fields.Update
    End With
    With oAppointment
        Set .Configuration = oConfiguration
        .StartTime = #3/22/1999 10:00:00 AM#
        .EndTime = #3/22/1999 11:00:00 AM#
        '.Attendees.Add ("Peterlam@Microsoft.com")
        .Location = "Atrium Building 2"
        .Subject = "Test Calendar Appointment"
        .Sensitivity = cdoCompanyConfidential
       'Add an attendee
        Set oAttendee = .Attendees.Add
        oAttendee.Address = "MAILTO:peterlam@microsoft.com"
        oAttendee.Role = cdoRequiredParticipant
       'To send an appointment request must create Calendarmessage object
        Set oCalendarMessage = .CreateRequest
        oCalendarMessage.Message.Send
    End With
End Sub
```

Code Sample 8- Sending a appointment to another user

Sending Multiple meeting requests in one Message

The new CDO object model makes it possible to send multiple meeting requests in one message. Individual meeting requests are held in the CalendarParts collections of the CalendarMessage object. Historically multiple messages would be sent to users for individual processing or a single message containing multiple appointments embedded which could eb dragged into the calendar in the Outlook client. The message MIME hierarchy can be seen in the CalendarParts collection. The example below shows the code used to create multiple meeting requests in one message.

```
'********************************************************************
' PROCEDURE    - CreateMultipleAppointments
' PARAMETERS   - None
' RETURN VALUE - None
' EXTERNALS    - Microsoft CDO library for Windows 2000
' DESCRIPTION  - Sends a message that contains multiple meeting requests.
'********************************************************************
 Sub CreateMultipleAppointments()
   'Procedure level variables
    Dim oAppointment1     As New Appointment
    Dim oAppointment2     As New Appointment
    Dim oCalendarMessage1 As CalendarMessage
    Dim oCalendarMessage2 As CalendarMessage
    Dim oConfiguration    As New Configuration
    Dim oAttendee         As New Attendee
   'Set the first appointment properties
```

1.

```
    With oAppointment1
        .StartTime = #3/29/1999 1:00:00 PM#
        .EndTime = #3/29/1999 2:00:00 PM#
        .Subject = "First Appointment"
        .Location = "Building B, Room 12"
       'Set the configuration fields
        oConfiguration.Fields(cdoSendEmailAddress) = "user1@microsoft.com"
        oConfiguration.Fields("CalendarLocation") = _
"exstore://exsvr03/private/user1/calendar"
        oConfiguration.Fields.Update
        Set .Configuration = oConfiguration
       'Add a required attendee
        Set oAttendee = .Attendees.Add
        oAttendee.Address = "MAILTO:user2@microsoft.com"
        oAttendee.Role = cdoRequiredParticipant
       'Create the calendar message
        Set oCalendarMessage1 = .CreateRequest
    End With
   'Create the second appointment item
    With oAppointment2
        .StartTime = #3/30/1999 2:30:00 PM#
        .EndTime = #3/30/1999 3:30:00 PM#
        .Subject = "Second Appointment"
        .Location = "Building B, Room 7"
        Set .Configuration = oConfiguration
       'Add the attendee to the second appointment
        Set oAttendee = .Attendees.Add
        oAttendee.Address = "MAILTO:user2@microsoft.com"
        oAttendee.Role = cdoRequiredParticipant
       'Add the second appointment to the calendar message
        Set oCalendarMessage2 = .CreateRequest
    End With

    oCalendarMessage1.CalendarParts.Add oAppointment2, _
cdoComponentTypeAppointment

   'Send the message
    oCalendarMessage1.Message.Send

 End Sub
```

Code Sample 9 - Sending multiple appointments in one message

An application that receives a message containing multiple meeting requests can extract each individual request by iterating through the CalendarParts of the message. Once a Bodypart has been found the appointment can be extracted by using the GetAccosciatedItem method. It is important to note at this point that the Outlook 2000 client can't process multiple calendar body part appointments.

Recurring Appointments

Users of Microsoft Outlook will be familiar with recurring appointments. They are appointments that occur on a given schedule such as monthly team meeting the first Monday of the month at 09:00. The principle for creating recurring appointments is the same as one off ad-hoc appointments, a standard CDO appointment item is created and

1.

then a RecurrencePattern object is applied to the appointment. The Recurrence object exposes properties that enable the developer to specify the interval, number of occurrences of the meeting and the date on which the appointments should end. Recurring appointments are very useful but it is important to be able to specify exceptions to the rule. The Exception object enables developers to add or delete specific instances of the recurring appointment, there is no limitation to the number of exceptions that are applied to a recurrence pattern but if there are a lot then developers should consider using standard appointment objects.

The code sample below shows how an appointment object can be made into a monthly recurring appointment but it caters for the fact that on July 5th there is national holiday. The recipients of the appointment would have their calendars updated on accepting this meeting. The sample assumes the start and end time of the appointment have already been set when they are passed to the procedure. The appointments can be updated by the meeting organizer in the future using the CDO object model.

```
Private Sub MakeAppointmentReccur(oAppointment as CDO.Appointment)
   'Procedure level Variables
    Dim oRecurrence      As CDO.IRecurrencePattern
    Dim oExceptionDate   as CDO.IException
    Dim oCalendarMessage As CDO.CalendarMessage
    Dim oAttendee        As New CDO.Attendee
   'Create the RecurrencePattern object
    Set oRecurrence = oAppointment.RecurrencePatterns.Add("Add")
   'Define the recurrence pattern
    With oRecurrence
        .Frequency = cdoMonthly
        .Interval = 1  'every 1 Month
        .PatternEndDate = #11/29/99 09:30:00 AM#  'the last appointment
    End With
   'Delete the appointment instance on July 5 as it is a national holiday
    Set oExceptionDate = oAppointment.Exceptions.Add("Delete")
    oExceptionDate.RecurrenceID = #7/5/1999 10:00:00 AM#
   'Add an attendee
    Set oAttendee = oAppointment.Attendees.Add
    oAttendee.Address = "MAILTO:peterlam@microsoft.com"
    oAttendee.Role = cdoRequiredParticipant
   'Create the calendar message and send it
    Set oCalendarMessage = oAppointment.CreateRequest
    oCalendarMessage.Message.Send
End Sub
```

Code Sample 10 - Setting recurring appointments with exceptions

1.

Contacts in CDO

The CDO object model has now incorporated some new functionality to deal with contacts. Outlook users are familiar with contacts but to CDO this a new area. Contacts can be one of three types, a person, a group or an organisation. There are interfaces for each of these classes of contact namely, IPerson, IGroup and IOrganisation. The Group contacts will replace the need for the personal address lists that users used in CDO under Exchange 5.5. The contact information is held in the VCard/MIME format which is an internet format outlined in RFC 2426.

The code sample below shows that a CDO Person object is created in order to obtain the URL of that person's contacts folder. This is needed so that when the new contact information is saved it is created in the correct mailbox.

```
'**************************************************************************
' PROCEDURE    - CreateContact
' PARAMETERS   - None
' RETURN VALUE - None
' EXTERNALS    - Microsoft CDO library for Windows 2000
' DESCRIPTION  - Creates a new contact in the Exchange Store
'**************************************************************************
 Private Sub CreateContact()
   'Procedure level variables
    Dim oPerson         As New CDO.Person
    Dim oMailBox        As CDO.IMailbox
    Dim oContactsFolder As CDO.Folder
    Dim sURL            As String
   'Get an object reference to the person who will own the contact
    oPerson.DataSource.Open ("http://fincollab/Exchange/peterlam")
   'Get URL path to the current users contacts Folder.
    Set oMailBox = oPerson
    Set oContactsFolder = oMailBox.Contacts
    sURL = oContactsFolder.DataSource.SourceURL
   'Set the properties of the contact item
    With oPerson
        .NameSuffix = "Mr."
        .FirstName = "Terry"
        .LastName = "Lamsdale"
        .HomeCity = "Wolverhampton"
        .HomePhone = "01902 567267"
        .WorkPhone = "01902 754541"
        .Email = "mailto://TerryL@ComptonRidge.com"
        .Fields("Organization") = "Compton Ridge Ltd"
        .Fields("OrganizationURI") = "http://www.ComptonRidge.com"
       'Commit the Contact item to the Exchange Store
        .DataSource.SaveTo sURL
    End With
   'Close down
    Set oMailBox = Nothing
    Set oPerson = Nothing
 End Sub
```

Code Sample 11 - Creating a contact for a person

An Introduction to OLE DB and ADO

Many developers are familiar with ODBC but don't understand why OLE DB is the preferred method of accessing data. ODBC is designed to provide access primarily to SQL data in a multi-platform environment. OLE DB is designed to provide access to all types of data in an COM environment. OLE DB includes the SQL functionality defined in ODBC but also defines interfaces suitable for gaining access to data other than tabular SQL data.

ActiveX Data Objects (ADO) is a COM automation server that exposes a high-level set of database programming objects to developers on top of an OLE DB data source. OLE DB is accessed by C++ programmers whereas ADO is accessible from all Microsoft Developer Studio tools and Microsoft Office. The two interfaces provide all developer communities with a suitable level of access to information in the Exchange store. Independent Software Vendors (ISV) often require very low level access to the data through OLE DB whereas most corporate developers work with ADO due to the simplicity of the interface.

These interfaces offer excellent access to data in tabular formats such as Microsoft SQL Server and other ODBC/ISAM databases but Exchange Server information is held in a variety of structures including documents. The 'OLE DB for docs' interface enables developers to gain access to stream data in structured storage documents and encoded document schemas such as MIME based contents like iCalendar and VCard.

Data can come in many forms and not all of them can be represented in tabular format. Figure 2 below shows how data might be structured in a document for a calendar entry in a custom application.

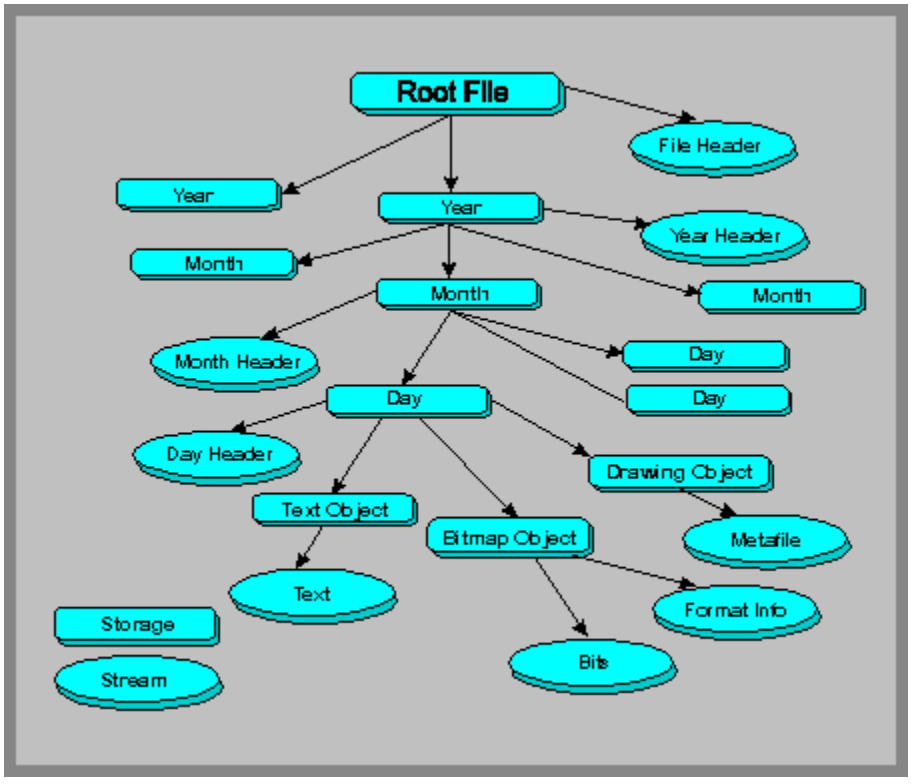

Figure 2 - How a custom application might store a calendar entry

OLEDB for Docs provides a mechanism for accessing structured non-tabular data. Microsoft has built its universal data access methodology around OLEDB and it is therefore the preferred choose for accessing information held in the “Platinum” store. The figure below shows how ADO is used to Access “Platinum” Server information stores using the same architecture as is used for other data formats. Many developers already have skills in OLEDB and ADO so they will find the transition to building “Platinum” collaborative applications much easier.

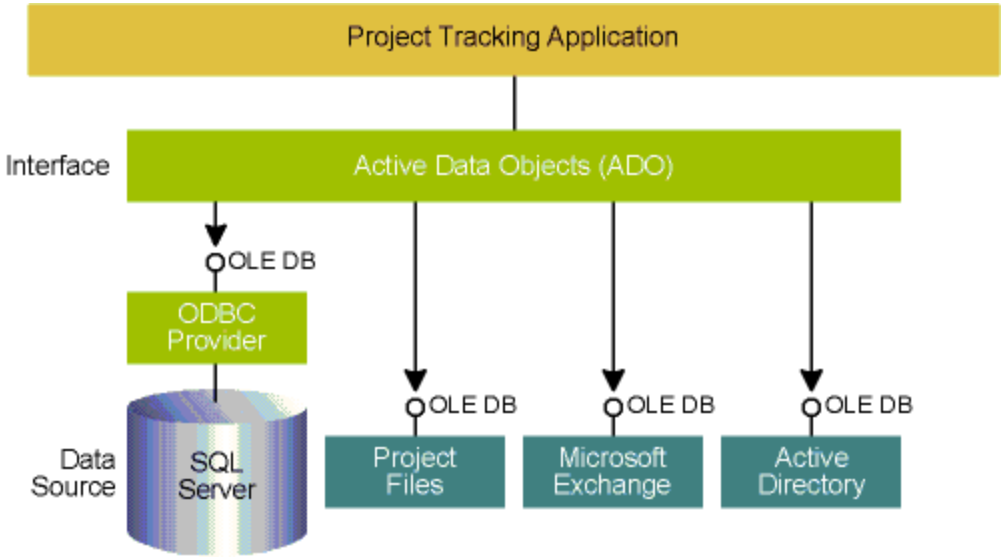

Figure 3 - OLE DB/ADO Universal data Access

With OLEDB and ADO access to the data in the “Platinum” store will be much simpler as all items in the database will have a documented schema to make forming SQL queries much simpler. The properties in the underlying content stream will be promoted up to the database and be kept fully synchronized when changes are made at any of the layers i.e. content stream, MIME body parts, OLEDB or high level CDO objects.

Creating reports on the “Platinum” store will be a simple as creating an SQL Query. CDO in “Platinum” will expose public/private stores as database tables as oppose to hierarchical file system approach of CDO 1.21, this fits well with Internet protocols such as NNTP and IMAP which presume a flat list of exposed databases.

Accessing a mailbox

The integration between ADO and CDO provides a more complete programming model. The code sample below uses the new URL technique of CDO to get an object reference to a mailbox. Access to each item within the inbox is then provided by calling the *GetChildren* method of the folder object and each item in that users Inbox is seen as an ADO record. This routine simple prints out the fields associated with the message such as RESOURCE_STREAMSIZE, RESOURCE_PARENTNAME etc. These are not the fields from within the message contents. In order to access these fields the individual messages need to be opened, an example of this is shown in *Code sample 11.*

1.

```
'*********************************************************
'PROCEDURE    - GetItemsInMailbox
'PARAMETERS   - None
'RETURN VALUE - None
'EXTERNALS    - Microsoft CDO library for Windows 2000
'              Microsoft ActiveX Data objects Library 2.5
'DESCRIPTION  - This Procedure works through the Inbox of
'              the user peterlam and displays the field
'              information associated with each item.
'*********************************************************
 Private Sub GetItemsInMailbox()
   'Procedure level variables
    Dim oFolder    As New ADODB.Record
    Dim oItems     As ADODB.Recordset
    Dim oField     As ADODB.Field
    Dim sFolderURL As String
    Dim iLoop      As Integer

   'Set up error trapping for this routine
    On Error GoTo TRP_GetMailboxDataWithADO:

   'Get an object reference to a “Platinum” path
    sFolderURL = "http://fincollab/Exchange/peterlam/inbox/"
    oFolder.Open sFolderURL
    Set oItems = oFolder.GetChildren
   'Iterate through the items in the folder
    Do While Not oItems.EOF
        Debug.Print "***** New Item *****"
       'Iterate through fields in the item
        For Each oField In oItems.Fields
            Debug.Print oField.Name & ": " & oField.Value
        Next oField
        iLoop = iLoop + 1
        oItems.MoveNext
    Loop

   'Prevent code entering the error trap on successful run of code
    Exit Sub

TRP_GetMailboxDataWithADO:
    MsgBox prompt:=Err.Number & ": " & Err.Description, Title:="PML Error
Warning"
 End Sub
```

Code Sample 12 - List all items in a mailbox by using ADO

Accessing message contents

The message items are records within the “Platinum” store. To gain access to the contents of a mail message the object must be opened. The CDO.Message object exposes a IDatasource interface which has an open method that takes a ‘Source’ parameter which expects an Exchange store URL. The value that should be passed into the source

1.

parameter is the text returned in the RESOURCE_ABSOLUTEPARSENAME field from the item in the “Platinum” store. The format of this string is the path of the object and the subject of the message encoded to use the characters ‘%20’ instead of spaces. An example would be

http://fincollab/Exchange/peterlam/Inbox/CDO%20is%20great

The code sample below is a procedure that takes a message object and prints all of the contents of the message into the debug window.

```
'*************************************************************
'PROCEDURE    - GetMessageFields
'PARAMETERS   - sMessageSource as string, this is a “Platinum”
'               URL that represents an item in a folder.
'RETURN VALUE - None
'EXTERNALS    - Microsoft CDO library for Windows 2000
'               Microsoft ActiveX Data objects Library 2.5
'DESCRIPTION  - This Procedure receives the Source URL of a
'               message item based on the value of the
'               Field "RESOURCE_ABSOLUTEPARSENAME" of an item
'               within a “Platinum” Folder. It prints out the name
'               and value of each field in the message including
'               the full text/HTML Body of the message.
'*************************************************************
Private Sub GetMessageFields(sMessageSource As String)
   'Procedure level variables
    Dim oMessage As New CDO.Message
    Dim oField   As ADODB.Field
   'Open a message using its source URL
    oMessage.DataSource.Open sMessageSource
   'Iterate the fields collection of the message object
    For Each oField In oMessage.Fields
        Debug.Print oField.Name & ": " & oField.Value
    Next oField
   'Close down the object references
    Set oField = Nothing
    Set oMessage = Nothing
End Sub
```

Code Sample 13- How to extract the fields form a CDO Message

Getting the raw stream content of messages

Streams provide developers with access to the raw content stored in objects. The use of ADO 2.5 enables the “Platinum” developer to make use of streams of data. It is possible to get the entire contents of a message in serialized format by using the GetStream method of a Message object. The message can be retrieved in either encoded or decoded formats to provide the developer with the maximum flexibility. The GetStream method uses the ‘MIMEFormatted’ property of the Message object to determine the format of encoding it will use. If the property is set to true the message is MIME formatted else the message is formatted in UUENCODE format.

1.

The example below shows a procedure that takes a parameter of a “Platinum” URL to a message and extracts the entire message as a stream into a file on disk. Streams can be saved or added to other messages as long as they don’t affect the structure of the host message.

```
'*********************************************************
'PROCEDURE    - GetMessageStream
'PARAMETERS   - sMessageSource as string, this is a “Platinum”
'               URL that represents an item in a folder.
'RETURN VALUE - None
'EXTERNALS    - Microsoft CDO library for Windows 2000
'               Microsoft ActiveX Data objects Library 2.5
'DESCRIPTION  - This Procedure receives the Source URL of a
'               message item based on the value of the
'               Field "RESOURCE_ABSOLUTEPARSENAME" of an item
'               within a “Platinum” Folder. It extracts the raw
'               stream of the message and saves it to a File.
'               This way it is possible to see the raw Mime
'               codes in its encoded form.
'*********************************************************
Private Sub GetMessageStream(sMessageSource As String)
   'Procedure level variables
    Dim oMessage As New CDO.Message
    Dim oStream  As New ADODB.Stream
   'Open the stream object
    oStream.Open
   'Open a message using its source URL
    oMessage.DataSource.Open sMessageSource
    oMessage.GetStream.CopyTo oStream
    oStream.SaveToFile "c:\streamdump.txt", adSaveCreateOverWrite
   'Close down the object references
    Set oStream = Nothing
    Set oMessage = Nothing
End Sub
```

Code Sample 14 - Extracting the raw stream of a message with ADO

Using the Bodyparts collection of a message it is possible to be more granular in extracting streams from a message. The Bodypart object exposes a GetDecodedContentStream and GetEncodedContentStream which enable developers to extract the particular portion of message in the format they require. The GetStream method returns the entire message.

OLE DB Providers for Exchange

The Exchange store is accessed using OLE DB and ADO though two OLE DB providers provided by Exchange. These OLE DB providers can be used from Microsoft Window 9.X and Windows NT 4.0 and later operating systems. There are two OLE DB providers, Exchange OLE DB for local use, and Microsoft OLE DB Provider for Internet Publishing (formerly known as Rosebud). Developers don’t need to make conscious decisions about which OLE DB provider they use as Exchange determines the appropriate provider to use

1.

given the protocol that is being used to access the Exchange Store. Remember that data access is done via URLs, these URLs come in two forms

- URLs using "http://" use the Internet Publishing OLE DB Provider
  `E.g. http://myserver/exchange/peterlam/inbox/`

- URLs using "file://" use the Exchange OLE DB Provider.
  `E.g. "File://./backofficestorage/Microsoft.com/MBX/peterlam/inbox/"`

1.

CDO integration with ADSI

Active Directory Services Interface (ADSI) is a COM interface that enables developers to interact with directory services such as Lightweight Directory Access Protocol (LDAP) which is supported by “Platinum”. You can use ADSI with its LDAP provider to manipulate any object in the Windows 2000 Active Directory. ADSI includes an OLE DB provider for rich querying of the directory namespaces from OLE DB directly or through ADO which enables developers to perform faster queries on the directory such as finding all users with the name ‘Peter’ in the ‘MCS’ department. The ADSI interface has been more closely coupled with the new versions of CDO to provide closer integration between the two programming models. A good example of this is the ability to create a new mailbox. The code sample below shows how the integration is closely coupled between CDO and ADSI.

```
Sub CreateMailbox()
   'Procedure level variables
    Dim oExchangeOrg As ActiveDs.IADSContainer
    Dim oUser        As ActiveDs.IADsUser
    Dim oPerson      As CDO.person
   'Create Reference to the Exchange organistaion to add a new user
    Set oExchangeOrg = GetObject("ldap://OU=Exchange, DC=PML, DC=com")
   'Create a new user
    Set oUser = oExchangeOrg.Create("user", "cn=MMouse")
   'Set LDAP properties of the newly created user
    With oUser
        .Put "saMAccountName", "MMouse"
        .Put "userPrincipalName", "MMouse"
    End With
   'Set EMO attributes
    With oPerson
        .DataSource.BindToObject oUser, "IADsUser"
        .mailboxAdmin.storeQuota = 800
        .mailboxAdmin.MailDrop = "http://Larry/private"
        .DataSource.save
    End With
   'Commit the object to the Active Directory
    oUser.SetInfo

End Sub
```

Check this code works

Protocol Events

The SMTP and NNTP services supplied in Microsoft Windows 2000 expose a COM architecture that enables developers to use trap default events and add their own custom events. Events are fired when a message is either sent to or from the service, such as a message being sent or arriving in the pickup directory. The develop can capture these events in code and add custom functionality to the protocol that provides them. The events work in a synchronous fashion in that when a message is passed to the event nothing else will happen to it until the code has finished with the message. This ensures that the message can't be accessed by other processes whilst the custom event code is executing.

These transport event sinks can be implemented by any code that implements the appropriate COM interface for the event. By implementing the interface the events are automatically processed by the application. These events called 'sinks' in COM. The sinks can be programmed using any Microsoft Developer Studio tool and also from script languages such as VBScript or Jscript.

The ability to intercept messages opens up many possibilities for developers. Below is a list of example uses;

- Virus checking on inbound/outbound messages.
- Redirecting messages from their original delivery path or re-addressing messages to alternative users. For example messages coming into Enquiry@Microsoft.com could be redirected to specific users, or stopping messages going to everyone in the business when this is not appropriate.
- Preventing mail messages from a specific originator or originators reaching the mail system. This would be useful to detect bulk message attacks or mailing list junk mail.
- Archival of messages.
- Posting messages to newsgroups that arrive via the SMTP service or vice-versa

The table bellows outlines the terminology used in the event architecture.

Term	Definition
Event source	The process that generated the event
Event sink	COM object that implements an event interface to service the event
Synchronous event source	An event source that calls the event sinks as the event condition occurs. It is assumed that multiple event sinks waiting on a single event will be called one at a time in the order dictated by the individual sink's sink priority.
Synchronous event sink	An event sink that either blocks or uses the calling thread while it processes the event.

	Event sinks that return or use a notify interface are considered to be synchronous event sinks because the source is required to wait for the notification before passing the event onto the next sink.
Sink priority	A hexadecimal number in the range 0x00000000 to 0xffffffff that dictates the order in which a sink is fired relative to the sink priority of all other sinks waiting for the same event.
Binding	The relationship between the source and the sink. e.g. this source will call this sink when the event occurs and these conditions are met

Applications that use protocol events

It is possible to create an application that responds to protocol events in any COM-based language, the example below is a Visual Basic implementation. The Microsoft Visual Basic Implements commands provide the hook for the interface and the four events that are exposed can be accessed. The message that the protocol receives is passed in as a parameter to all events and the standard CDO object model can be used to manipulate the message.

```
Implements ISMTPOnArrival
Implements INNTPOnPost
Implements INNTPOnPostFinal
'General –Declarations section of the module

Public Sub ISMTPOnArrival_OnArrival(ByVal objCdoMsg As CDO.Message)
   'Your Code here
End sub
Public Sub INNTPOnPost_OnPost(ByVal objCdoMsg As CDO.Message)
   'Your Code here
End sub
Public Sub INNTPOnPostFinal_OnPostFinal(ByVal objCdoMsg As CDO.Message)
   'Your Code here
End sub
```

Code Sample 15 - Using protocol event from Visual Basic

Managing the Events

It is possible to create several applications that use these events so information about event sources and sink bindings are stored in the binding database used by the Windows 2000 platform. A binding is the association of a particular event handler and a sink. Each

binding between the source and a sink is uniquely identified with a Global Unique Identifier (GUID) and a human friendly name. In the event of two applications wanting to catch the event the priority field from the database is used to determine which application will receive the event first.

Developers or system administrators can manage the event bindings database by using the Server Extension Objects (SEO) which provide programmatic access to the bindings table. CDO provides two command-line Windows Scripting Host (WSH) scripts, smtpreg.vbs and nntpreg.vbs, to simplify the management of protocol event bindings.

Below is some sample syntax that registers new bindings and then disables them

```
cscript smtpreg.vbs /add 1 onarrival Sink1 MySinks.MySink1  "mail from=*"
cscript nntpreg.vbs /add 1 onpost    Sink2 MySinks.MySink2 "newsgroups=*"
cscript smtpreg.vbs /disable 1 onarrival Sink1
cscript nntpreg.vbs /disable 1 onpost Sink2
```

Code Sample 16 - Using WSH to manage protocol event bindings

These scripts directly affect the bindings database. Below shows a representation of the protocol events bindings table.

ID	Display Name	SinkClass	Source Property Bag
{GUID1}	Sink 1	MySinks.MySink1	Priority = 150 Rule="mail from=*@microsoft.com"
{GUID2}	Sink 2	MySinks.MySink2	Rule="EHLO=*microsoft.com"
{GUID3}	Sink 3	MySinks.MySink3	Priority = 30000 Rule="rcpt to=Peter*;mail from=D*"

Figure 4 - The protocol events bindings table

It is noticed that there is the ability to specify the COM component that will call the event in the ‘Sink Class’ field and that the ‘Source property bag’ field allows for the provision of rules and priority numbers to be applied to a particular database entry.

Setting protocol event Rules and Priorities

In large organisations it is possible to end up with many applications that wish to manipulate messages at protocol level. If all events fired when a message went through the protocol it could take a very long time to process so it is possible to specify rules which define when an application should be notified of the event. This efficient process means that if the rule is not true the event isn’t passed to the application. In the table above the first entry in the configuration table specifies that the event should only be passed to the ‘MySinks.MySink1’ interface if the message is from someone at Microsoft. It

1.

is possible to create multi-conditioned rules by placing a semi-colon between the rules as shown in the table above for GUID3.

The events are sent to the sink classes that request the event one by one in serialized fashion in the order specified in the priority field. Priority values can range from 0 to 32767, with 0 being the highest priority and 32767 the lowest priority. The *default* priority is 2457. Sinks with higher priority always run before those with lower priorities and if two sinks are given the same priority number then they are run in the order chosen by the system.

In order to change the properties of a particular event binding use the aforementioned WSH scripts. The scripts take parameters that enable developers or system administrators to manipulate the settings of the protocol events.

```
cscript smtpreg.vbs /setprop 1 onarrival MySink1 Source Priority 100
cscript smtpreg.vbs /setprop 1 onpost    MySink2 Source Priority 30000
```

* A complete listing of the fields that can be set through the script are available in the Platform SDK.

Once the sink completes, the event status flag returned by the sink is checked to determine whether or not to continue notifying subsequent sinks. If the sink returns 'CdoEventStatus.cdoRunNextSink' for the event status, it proceeds to notify the next sink in the configuration database. If the sink returns an event status of 'CdoEventStatus.cdoSkipRemainingSinks', no further sinks are notified of the event.

Development considerations

- The events run in the NT security context of the 'Local System Account' although by using the Win32 API or Microsoft Transaction Server (MTS) it is possible to make the sink run under a different context.
- In a high volume mail system with many messages being created and being processed by event based applications the system will be doing a lot of additional work. It is possible to cache the Sink Classes by implementing the IEventIsCacheable interface, this enables the system to keep an instance of the sink class in memory and re-use it for subsequent events as oppose to creating a new instance each time the sink class needs to be used. Below is a code sample showing the code developers must add to achieve this.

```
Implements IEventIsCacheable

Private Sub IEventIsCacheable_IsCacheable()
  ' just return S_OK which means do nothing!
  ' Object will be cached for subsequent use
End Sub
```

- When an SMTP message is received by a protocol event the 'EnvelopeFields' collection of a CDO Message object is populated. The fields in this collection are documented in the http://schemas.microsoft.com/cdo/smtpenvelope Namespace. The fields include information like arrival time, the IP address of the client, sender address and the recipient list. It is also possible to change the message status of

the message being processed by the protocol event to specify whether the message should not be delivered or should be sent to the badmail directory.

1.

Exchange Store events

The OLE DB provider for Exchange Server and the “Platinum” stores own native event source infrastructure provide developers with a mechanism to capture events in custom applications. These events include the creation, deletion, retrieval or moving of Exchange data. The exposure of these events make the creation of workflow applications much simpler on Exchange. The mechanism works in the same way that the protocol events in the previous chapter in that the event sink COM applications are written and then bound to the event source via a scripted process. The store provider and event source infrastructure are encapsulated within the COM server ExOLEDB.DLL .

Store Event sinks are bound to store events by registering them using the Server Extension Objects (SEO). The ExOLEDB event provider reads the event registration database and translates the binding information into store ‘restrictions’. Whenever a store restriction is met, the ExOLEDB event provider triggers an event and calls the appropriate bound event sinks: The diagram below shows the architecture of the store events. It is noticed that the sinks are run out-of-process of the store thread which provides resilience from sink applications that crash.

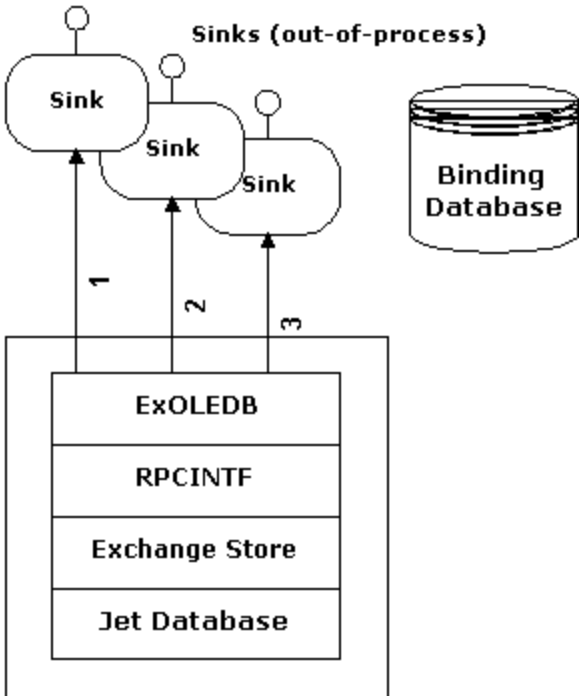

Figure 5 - The Exchange Store Events architecture

“Platinum” Store Event behavior

The protocol events do not expose as many events as the OLE DB and store events and those that are exposed execute synchronously. The OLE DB events can also execute in an asynchronous fashion. The Exchange OLE DB Store provider sources synchronous and asynchronous store events, as well a set of system events. The store events are

supported for both public folders and private mailboxes. System events are supported across the servers and store databases. The System events include; OnTimer, OnMDBStartUp, OnMDBShutdown, OnServerStartIp and OnServerShutdown. A complete listing of the fields that can be set through the script are available in the Platform SDK.

Once the sink completes, the event status flag returned by the sink is checked to determine whether or not to continue notifying subsequent sinks. If the sink returns 'cdoEventStatus.cdoRunNextSink' it proceeds to notify the next sink in the configuration database for that event. If the sink returns an event status of 'cdoEventStatus.cdoSkipRemainingSinks', no further sinks are notified of the event. This requires that developers who create sink classes must understand that their sink class must work cooperatively with other sink classes and error trapping must be very precise.

Asynchronous Event Sources

An event source that calls the event sinks after the event condition has occurred. Asynchronous event sources will not support the ability to abort the event because it has, by definition, already occurred. Asynchronous event sources do not guarantee that the event will be delivered in the event that the source crashes or is shut down before the sink has been called.

Reliable Asynchronous Event Sources

An event source that calls the event sinks after the event condition has occurred. Reliable asynchronous event sources guarantee that the sink will be called at least once. A reliable asynchronous event sink may also guarantee that it finishes processing the event by returning S_PENDING and notifying the source when it has processed the event.

Event scope

The scope of a event is the range in the store on which the event is applied. An event can be registered such that it applies to a given folder, or to a folder and all subfolders of that folder. The latter is referred to as a recursive event. Many sinks can a call the same event and is should be noted that if a sink is registered both as a recursive sink, and again within the recursive scope as a non-recursive sink, then the same sink will be called twice – once for the root and once for the interior node.

The "Platinum" release of Exchange Server has provided the ability for system administrators to have multiple databases on a single server. It is important to remember that the Exchange store does not provide support for sinks configured across databases I.e. if a single store machine has multiple databases, then sinks must be configured on each and every database. Similarly, the system will not perform any replication of sink information across Exchange stores.

All sinks run in the security context of the EUSER_SEOEVENT user. The Event provider impersonates the logon account as required and normal Exchange security is not compromised. If there are any errors in the event firing process then the Exchange OLE DB provider reports into the Windows 2000 event log. Event sinks should incorporate their own custom error handling.

Registering an Event

The registration process is very similar to the protocol events in that a WSH script is provided to add or remove event bindings. Write access to the event registration database [the metabase] is limited to the EUSER_EXEVENT_ADMIN group. This means

that only members of the EUSER_EXEVENT_ADMIN may register for SEO events. The EUSER_EXEVENT_ADMIN group is created during setup of Exchange. Below is an example showing how to use the WSH script to add a store event. The code shows that the event sink ‘PMLSink.MySink” will be called when the OnSave and OnDelete events fire on the inbox of the user StuartRi. The priority is set to high at 63 (Hexadecimal 0x3f).

```
CSCRIPT eventadmin.vbs add "onsave;ondelete" PMLSink.MySink _
        file://./exchangemailboxes/foo.microsoft.com/StuartRi/inbox -p _
        0x3f -m deep
```

Code Sample 17- Adding a store event to the metabase

For a more detailed breakdown of the Server store events registration process see Appendix A of this document and the CDO SDK.

1.

CDO System Management Objects (aka EMO)

The enhancements to the CDO Object model and the OLE DB integration has provided developers with a comprehensive tool for creating messaging, calendaring, contact management and data retrieval applications but none of these interfaces handle the management of Exchange itself. CDO system Management tools fills this role by providing a high level COM interface for programmatic administration of Exchange. In some preliminary "Platinum" documentation these objects were referred to as Exchange Management Objects (EMO).

These objects will enable developers to create custom management tools and ISVs can integrate Exchange administration into their tools[1]. Aswell as developers building feature rich applications it is also possible for system administrators to create scripts that will enable them to perform many tasks more simply such as creating a lot of new mailboxes or changing the storage quota for each mailbox very quickly. The Microsoft Management Console (MMC) snap-in for "Platinum" uses these management objects so developers should be confident that functionality exposed in the MMC interface can be automated.

CDO System Management enhances the CDO object model in *three* major areas

- It extends the CDO Folder manipulation abilities by providing a Folder administration interface
- It provides integration with CDO to create/delete/move/manipulate mailboxes and adjust setting for limits and protocols used by the Mailbox
- The stores, servers and sites will be exposed as objects.

CDO System Management and CDO both rely on OLE DB to access the underlying data sources in the Exchange store and Windows 2000 Active Directory.

Insert code examples for

1. *Iterating through all servers/sites in organisation*
2. *Update multiple Mailbox limits i.e. where name is 'Lamsdale'*
3. *Create/Delete Mailboxes*

```
Private Sub cmdGetAllServersInOrg_Click()
   'Procedure level variables
    Dim oOrganistaion As New EMO.Organisation
    With oOrganistaion
        .DataSource.Open ("LDAP://microsoft.com")
        .Sites.MoveFirst
        Do While Not .Servers.EOF
            'do something with each server
        Loop
    End With
```

[1] By final release it is expected that "Platinum"'s Microsoft Management Console (MMC) interface will use CDO System Management objects to manipulate Exchange objects.

```
End Sub
```

```
Private Sub cmdGetAllSitesInOrg_Click()
   'Procedure level variables
    Dim oSite As New EMO.Site
    With oSite
        .DataSource.Open ("LDAP://nyorkshire.com/dales")
        .Stores.MoveFirst
        Do While Not oSite.Stores.EOF
            'do something with each store
        Loop
    End With
End Sub
```

1.

The Exchange Schema

The Exchange Schema is a database of properties and objects organized by namespaces, encompassing all the properties for messaging, calendaring, contact management, knowledge management, and other Microsoft BackOffice technologies. The schema has been designed so that it can be accessed from intranet clients with HTTP calls using the Distributed Authoring and Versioning (DAV) extensions and the Microsoft OLE DB Provider for Internet Publishing. The Exchange schema is transmitted as XML elements that are parsed by ADO and CDO to extract real data from the Exchange store. There are a lot of separate namespaces which are listed below;

DAV:
MAIL:
http://schemas.microsoft.com/cdo/configuration/
http://schemas.microsoft.com/cdo/nntpenvelope/
http://schemas.microsoft.com/cdo/smtpenvelope/
http://schemas.microsoft.com/cdo/workflow/
http://schemas.microsoft.com/exchange/
http://schemas.microsoft.com/exchange/replication/
http://schemas.microsoft.com/hotmail/
http://schemas.microsoft.com/mapi/
http://schemas.microsoft.com/outlook/
http://schemas.microsoft.com/recipients/
http://schemas.microsoft.com/repl/
http://schemas.microsoft.com/windows/
http://www.bizprop.com/
urn:content-classes:
urn:schemas-microsoft-com:exch-data:
urn:schemas.microsoft.com:fulltextqueryinfo:
urn:schemas.microsoft.com:htmlinfo:
urn:schemas.microsoft.com:htmlinfo:linkinfo:
urn:schemas.microsoft.com:htmlinfo:metainfo:
urn:schemas.microsoft.com:htmlinfo:scriptinfo:
urn:schemas.microsoft.com:office:office:
urn:schemas:calendar:
urn:schemas:calendar:workendtime
urn:schemas:contacts:
urn:schemas:events:
urn:schemas:folder:
urn:schemas:httpmail:
urn:schemas:mailheader:
urn:schemas:properties:

Within each of these namespaces there are many individual properties that can be requested or modified. To gain access to a property the syntax is namespace:property. Developers will find that many of these properties are represented by the CDO object model which means they can avoid using the native syntax. The CDO Object model provides properties such as the ‘Subject’ property that actually represents ‘urn:schemas:mailheader:Subject’, but it is possible to use the namespace properties directly in code as shown *overleaf*. The Message object has a fields collection that can be accessed with ADO. The fields have an index value and a namespace name value. The field names are the properties of each namespace such as the ‘From’ property of the ‘urn:schemas:mailheader” namespace.

```
Dim oMessage as New CDO.Message
Dim oFields  as ADODB.Fields
Set oFields = oMessage.Fields
With oFields
  .Item("urn:schemas:mailheader:from") = "Someone@Microsoft.com"
  .Item("urn:schemas:mailheader:to") = "SomeoneElse@ Microsoft.com"
  .Item("urn:schemas:mailheader:subject") = "Your lights are on"
  .Item("urn:schemas:httpmail:textdescription") = "Turn them off!"
  .Update
End With
oMessage.Send
```

Code Sample 18 - Using Namespace properties directly in CDO

Note: The configuration examples shown previously in this document also used namespaces, more specifically the 'http://schemas.microsoft.com/cdo/configuration/' namespace.

Understanding DAV

DAV is important to Exchange because this is the way that client applications will get information form the Exchange store or the Active Directory. DAV is an extension to the HTTP 1.1 standard for exposing a hierarchical file storage media, such as a file system, relational database schema or hierarchically structured database such as the Exchange store, over an HTTP connection. DAV is typically used in scenarios where the data is stored on the server but used on the client.

HTTP is an extensible protocol that was originally a connectionless request / response protocol. The basic structure of a HTTP response is a verb (GET, PUT, POST), a requested Uniform Resource Locator (URL) with custom headers and a protocol name. Extending on HTTP has several advantages in that it is already widely adopted and offers support for Proxy firewalls, authentication, ubiquitous clients, offline usage and the HTTP Namespace. DAV will add new verbs to HTTP such as PROPFIND (retrieve property values from resources) and PROPPATCH (modify properties of resources) that enables requests to use Extensible Markup Language (XML) to express complex syntax for interacting with metadata resources.

The World Wide Web Consortium (W3C) specification for XML provides a namespace mechanism to prevent name collision. This aids XML syntax in that the full namespace can be represented as a prefix such as the XML shown below. The namespace is for Microsoft office but this demonstrates how a namespace can be used, 'O' becomes a short hand for the schema and the 'Author' and 'Title' properties are populated.

```
<O:author xmlns:O="urn:schemas.microsoft.com:office:office:">
  Lisa
</O:author>
<O:title>DAV at Microsoft -- Overview</O:title>
```

1.

It is important to note that, although the format of a namespace prefix resembles that of a URL, the namespace is not in fact a URL, and it may not correspond to any existing web page. It is fact a Document Type Definition (DTD) that contains properties that the namespace contains. It is precisely this mechanism that the Exchange store uses to expose the various properties of its own namespaces.

DAV Searching and Location (DASL) protocol.

The Internet Engineering Task Force (IETF) provides many drafts for up and coming technology and help to form the industry standards that Microsoft is conforming to. The IETF are currently working on the DAV Searching and Location (DASL) protocol.

The motivation for DASL is that WEBDAV and HTTP provide support for client-side search, but not server-side search. The HTTP GET method allows clients to retrieve a resource's content; the PROPFIND method defined in WEBDAV allows clients to retrieve a resource's properties. Having retrieved a resource's properties and/or content, the client can compare them to its search criteria to determine whether the resource is of interest. Although this client-side searching is logically sufficient, and requires no modifications to the server, it comes at a significant cost, because it makes inefficient use of network resources. A client must retrieve properties and content for each targeted resource, which does not take advantage of server intelligence. Servers capable of searching can use sophisticated mechanisms to generate results: internal caching of intermediate search results, content-indexing, etc. Client-side search limitations are exposed by queries such as "find all text files modified during the last week" . This problem is exacerbated when multiple clients are making similar queries, demonstrating scalability issues.

DASL allows for server-side searching. Server-side searching allows the client to formulate a query and have the server perform task of selecting the resources that fit the criteria. This overcomes the limitations of client-side searching described above. The benefit is a searching solution that scales.

For a full description of the internet draft on DASL. Please refer to the following URL.
http://www.ietf.org/internet-drafts/draft-ietf-dasl-requirements-00.txt

Querying the Exchange Store

The “Platinum” store needs to be easily accessible and searchable for developers of custom applications. The schema enables developers to understand the various properties of the Exchange store but doesn’t provide a searching mechanism. There is an OLE DB Provider (formerly known as Rosebud) for the “Platinum” store that excepts a syntax very similar to traditional Structured query Language (SQL). The SQL syntax is actually following the DASL specifications.

The code sample below shows how the search path is specified as the Inbox for the Exchange user peterlam. The path is passed to an ADO Connection object along with a reference to the OLE DB provider for Exchange. This OLE DB provider will enable the ADO Connection to create an object reference to the specified section of the “Platinum” store. The DASL specification enables queries to express ‘scope’ and ‘criteria’ and these can be passed to the OLE DB provider.

The query is processed on the server and then the client application will only receive the records that match the SQL criteria. It is noticed that the Exchange schema namespace properties are used in the SQL syntax not the user friendly names for these fields. The records are returned as a standard ADO Recordset and are then ready for manipulation.

1.

This ability to query the Exchange store using the same skills as accessing a SQL Server will mean developers will simply need to learn the Exchange Schema in order to port their skills to developing Exchange applications.

```
'*********************************************************
'PROCEDURE    - QueryMailboxContents
'PARAMETERS   - None
'RETURN VALUE - None
'EXTERNALS    - Microsoft CDO library for Windows 2000
'              Microsoft ActiveX Data objects Library 2.5
'DESCRIPTION  - This Procedure
'*********************************************************
 Private Sub QueryMailboxContents()
   'Procedure level variables
    Dim oConnection As New ADODB.Connection
    Dim oRecordset  As New ADODB.Recordset
    Dim sDataSource        As String
    Dim sFilter     As String
   'Specify the Exchange URL to be interrogated
    sDataSource = _
"file://./BackOfficeStorage/finance.mcs.microsoft.com/MBX/peterlam/Inbox"
   'Connect to data source through Exchange OLEDB provider
    With oConnection
        .ConnectionString = sDataSource
        .Provider = "ExOLEDB.Datasource"
        .Open
    End With
   'Specify the SQL filter for the data to be retrieved
    sFilter = " where ""urn:schemas:mailheader:subject"" <> 'FOOBAR'"
   'Retrieve the records from the “Platinum” Store
    With oRecordset
        .Source = "SELECT * from scope('shallow traversal of """ &
sDataSource & """')" & sFilter
        Set .ActiveConnection = oConnection
        .Open
    End With
   'Iterate through the items in the folder
    Do While Not oRecordset.EOF
        MsgBox oRecordset.Fields(0).Value
        oRecordset.MoveNext
    Loop
End Sub
```

Code Sample 19 - Using the Exchange Schema to query the store

The schema properties can be seen by going into the \EXCHSRVR\SCHEMA directory on the machine where Exchange is installed. The directory contains many XML files that contain the properties of the various namespaces. The namespaces are internally understood by Exchange but the properties of the schema are exposed via the XML files.

If developers iterate through the fields collection of a message they will see the name property of the field is the namespace property name. This could act as useful learning guide to developers wishing to get to grips with the way the schema maps onto messages created in client applications such as Microsoft Outlook.

1.

1.

Workflow Objects

There has been much discussion on the ability of Exchange Server to deliver workflow applications. It has traditionally been a developer intensive experience due to the lack of dedicated architecture. The “Platinum” release of Exchange Server will provide a dedicated workflow engine. The workflow engine will be implemented as a DLL, and developers will be able to create applications via a COM automation server component, CDO Workflow objects for Exchange DLL. This DLL will provide both design-time and run-time objects for creating, managing and using workflows.

The traditional view of workflow is that of a document that gets passed through a business process will various parties processing the document at each stage. The document process is tracked and exception handling such as time-outs or expiration are handled, by a workflow engine. In “Platinum” terminology the document is a record in a folder (what would have been referred to a message under MAPI).

Note The workflow engine is a huge advancement on the Exchange Server 5.5 Routing objects. These applications will not be migrated to the new architecture automatically. These workflow applications will need to be recreated or continue using the old architecture.

The Workflow Engine Architecture

The architecture for the workflow engine is shown in *Figure 6*. The process for interacting with items in the Exchange store is the same process for normal CDO operations in that client access will be via DAV and OLE DB.

In “Platinum” the ‘state’ information of a workflow item will be stored in the item itself, there will be no external database holding this information. The state information will not be made public and will not be accessible to users through client interfaces or through code to ensure the integrity of the data. The “Platinum” per-property (MAPI Field) ACL security settings will enable this data hiding to be achieved. The workflow is designed so that any state changes in an item are made by the workflow engine not by the user, although the system will invariably make state changes based on user modifications to the workflow item.

For any given process there will be a number of given states that a workflow document can be in. The states are held in the ‘Action Table’. Action tables can be created or modified via code or by creating an XML document which can be imported. The format of the XML is outlined in the ‘XML Process Definition’ specification.

There are two levels of Action table

- **Folder Action Table.** In many cases the items in a folder will all follow the same business process and any new document created in the folder will automatically inherit the default folder action table.
- **Document Action Table.** This is an action table that is specific to a document. These can be one off ad-hoc action tables.

Each row in an Action table represents a transition from one state to another for the workflow process. The workflow engine exposes six events; as OnCreate, OnDelete, OnTimer, OnEnter, OnExit and OnTransition. These events will be exposed to developers who will be able to associate custom code with the events.

Developers can write script or COM objects that can be associated with these events, the workflow engine will fire these scripts as required through the script engine.

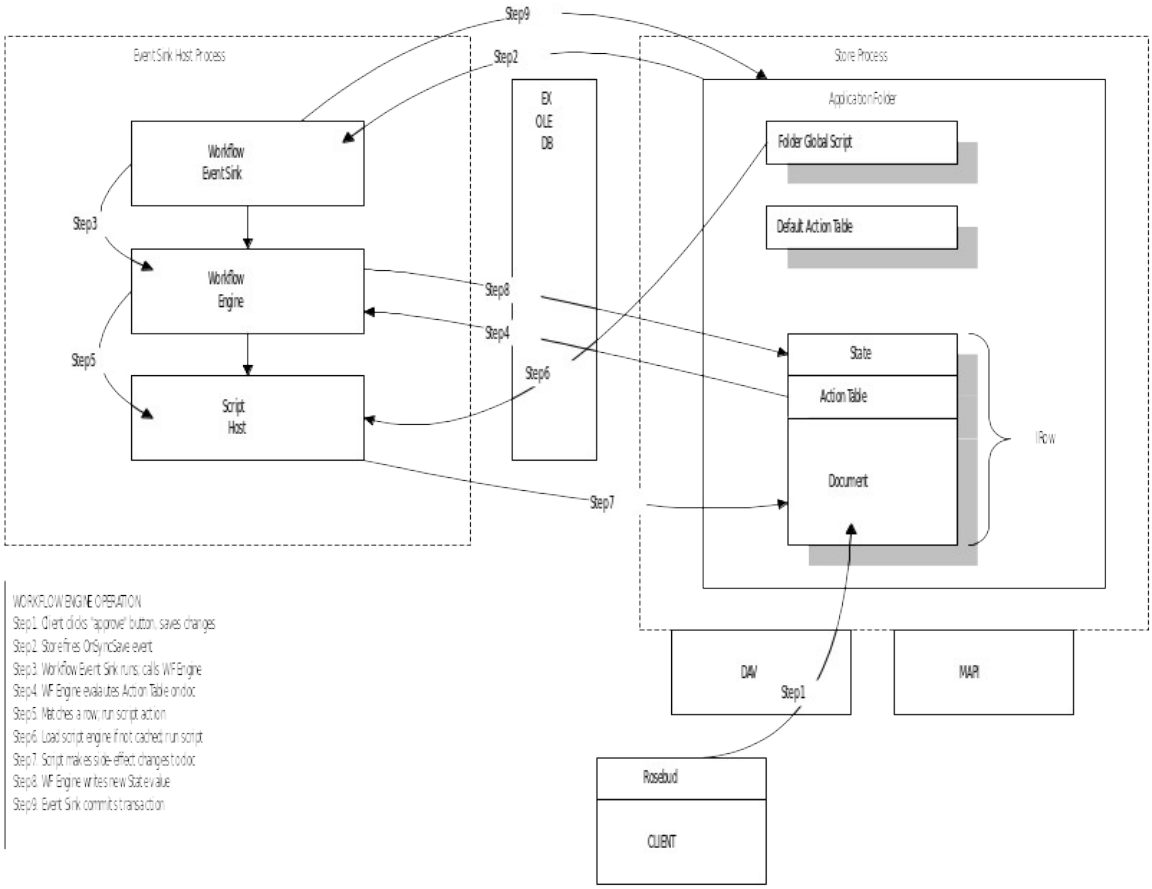

Figure 6 - Workflow Engine Architecture

Complex Workflow operations

There are predominantly two types of workflow, node to node (routing slips principles) or Hub and Spoke (a central point is used to action items). In simple form workflow is very straightforward to implement but there are times when multiple actions must be taken in parallel for an item to progress. This ‘branching’ behavior requires a work item to go through two parallel tracks and then merge the details of the process later. This process is often called Split-Join or Split-Rendezvous.

The “Platinum” store can implement versioning of documents using DAV and this enables the workflow engine to create an instance of the form for each concurrent parallel process. The document is essentially cloned and issued for processing. When the various processes have been carried out the documents will be merged back into one document.

Conclusions

1.

Appendices

Appendix 1 - Using Event Store Bindings

Exchange store events are defined for a particular store and folder. Use the eventadmin.vbs WSH script with the following parameters to add a sink binding for the following store events:

Asynchronous events: OnSave, OnCopy, OnMove and OnDelete

Synchronous Events: OnSyncSave, OnSyncDelete, OnSyncLock, OnSyncUnLock

The syntax of the WSH script is as below

```
cscript eventadmin.vbs Add
                            EventName
                            SinkClass
                            Scope
                            [-p Priority]
                            [-m MatchScope]
                            [-f CriteriaFilter]
                            [-s ScriptFilePath]
                            [-g StoreGuid]
                            [-guid BindingGuid]
```

EventName Parameter

The name of the store event for the binding. Should be one or more of the following: OnSave, OnDelete, OnCopy, OnMove, OnSyncSave, OnSyncDelete, OnSyncLock, or OnSyncUnlock. Case in the names is ignored. Multiple events can be specified for the same binding by placing semi-colons (;) between the names of each event. For example, "onsave;onmove."

SinkClass Parameter

The class identifier (in registry format) or programmatic identifier of the sink COM class. For example, "MySinkSink1."

Scope Parameter

The fully qualified URL to the folder for which the sink binding is intended. For example, file://backofficestorage/public%20folders/events.

Priority

[optional] The binding's relative priority. Values range from 0x0000 (max) to 0xFFFFFFFF (min). The default is 0xFFFF. Sinks with no explicit priority set default to 0xFFFF. Sinks with the same priority run in indeterminate order.

MatchScope

[optional] The scope for the binding. Can be one of DEEP | SHALLOW | EXACT | ANY. Bindings with no defined scope default to EXACT.

1.

CriteriaFilter

[optional] An SQL WHERE clause defining additional conditions that must be met by a triggering item for the sink to be notified of the event. For example,

```
WHERE CONTAINS("urn:schemas:mailheader:to","someone@somedomain.tld").
```

ScriptFilePath

[optional] The full path to the script file containing sink code. An event scripting host is used to run the script if this value is set. If no script is specified,

StoreGuid

[optional] The GUID identifying the store for which the binding is to be registered.

BindingGuid

[optional] The binding GUID to use when adding or modifying the binding. If no GUID is specified, one is created automatically for you and a new binding is added.

1.

Appendix 2 – Adding a System Event Binding

System events are defined only for a particular server or database and do not correspond to any particular folder or store. Use the eventadmin.vbs WSH script with the following parameters to add or modify a sink binding for the following store system events:

- OnServerStartUp
- OnServerShutDown
- OnMdbStartUp
- OnMdbShutDown

The syntax that the WSH script uses is as below

```
cscript eventadmin.vbs Add
                            SystemEvent
                            SinkClass
                            [-s ScriptFilePath]
```

SystemEvent

One of the 4 system events for the binding. Multiple events can be associated with the same binding by listing the events separated with semi-colons (;). For example, "onmdbstartup;onmdbshutdown." In this case, the bound sink would be notified of both the startup and the shutdown of any Exchange store database (mdb).

SinkClass

The class identifier (in registry format) or programmatic identifier of the sink COM class. For example, "MySinks.Sink" or {GUID}.

ScriptFilePath

This is an optional parameter that can be used if the sink class is implemented as a script file. The parameter value should include the full path to the script file containing sink code. An example of the syntax is shown below.

```
cscript eventadmin.vbs add onserverstartup sink1.sink1.1 -s _
c:\temp\script.vbs
```

There is also an OnTimer system event which can be set up to fire on the basis of time rather than a physical event occurring in the Exchange store. The same WSH script file is used as for the other system events but the syntax is different as shown below.

```
cscript eventadmin.vbs Add
                            OnTimer
                            SinkClass
                            StartTime
                            Interval
                            [-e ExpiryTime]
                            [-s ScriptFilePath]
```

1.

The StartTime parameter and ExpiryTime parameter are specified in the format of "8/4/98 01:50:00 AM " and the Interval parameter is specified in the number of minutes between each instance of the

event so a value of 60 would mean the event would occur every hour until the expiry time occurred. Below is a sample of how to set up a system OnTimer event.

```
cscript eventadmin.vbs add ontimer sink1.sink1.1 "8/4/98 01:50:00 AM" 1 -e
"8/4/98 02:25:00 AM"
```

1.